Paper No. 18

SIMULATION AS A COMPUTER DESIGN AID

J. E. Weatherbee and D. S. Taylor, *Computer Sciences Corporation, Huntsville, Alabama*
B. C. Hodges, *Computation Laboratory, Marshall Space Flight Center, Huntsville, Alabama*

ABSTRACT

The use of simulation in evaluating hardware and software design concepts for the Automatically Reconfigurable Modular Multiprocessing System (ARMMS) is discussed. Two simulation models are described and results obtained from simulations conducted with these models were presented.

INTRODUCTION

Future space missions such as the earth-orbiting space station, astronomical space observatory and numerous outer-planet space probes will be measured in years instead of days. Long range missions of this type will place increased requirements on the onboard digital computing equipment, particularly in the areas of reliability and computing capacity. The use of third generation computing concepts such as micro-programming and multiprocessing for computational speed and fault tolerant computing concepts such as modular component design, redundancy, and system reconfigurability has made the design of future spaceborne computer systems no longer the relatively simple task it once was. The Marshall Space Flight Center, Huntsville, Alabama, is currently designing a complex avionics computer for use in the late 1970's to mid 1980's time frame. This system, the Automatically Reconfigurable Modular Multiprocessor System (ARMMS) (1), utilizes many of the above mentioned concepts previously not available for spaceborne systems. The ARMMS design goal is to meet the increased reliability and computing requirements of future spaceborne computer systems through a highly modular computer architecture which can be configured as a multiprocessor for maximum computing speed and as a duplex or triple modular redundant (TMR) system with standby spares for extremely high reliability. Moreover, the ARMMS will be dynamic in that it will be possible to alter its configuration, in real time, as required by various mission phases or events.

One problem encountered by the ARMMS design group was to determine an effective means for measuring the effect upon system performance of the various design concepts being considered. Since some of these features (e.g., dynamic reconfiguration) are new not only to avionics computers, but to computer systems in general, there were no existing systems which could provide performance data relevant to the ARMMS design. This problem was resolved in part through the use of discrete event simulation, a technique which has gained general acceptance for use in the design and evaluation of computer systems. (2, 3)

Areas of interest for the application of simulation ranged from the instruction execution level to the simulation of the scheduling algorithms of the ARMMS executive system. Due to the high cost of computer time required to simulate all of these activities in a single model, two different models were developed; one which simulates the traffic between the central processing units (CPU's) and central memory (CM) as instructions are fetched and executed, and another which models the principal algorithms of the executive system required for resource allocation and task dispatching. This paper describes the two models and presents some results obtained from their application.

SYSTEM DESCRIPTION

The ARMMS design features a highly modular computer architecture which allows the system hardware to be configured according to the needs of individual tasks; highly critical tasks will be executed in triple modular redundant (TMR) or duplex mode while less critical tasks will be executed in simplex mode. Sufficient system components will be available to allow the concurrent execution of more than one task and to provide spare modules to replace certain critical system components in the case of failures. Thus the ARMMS executive must not only perform the functions required in a third generation multiprocessor system, but will also be charged with the tasks of system configuration control and response to component failures. A hardware/software complex, the Block Organizer and System Scheduler (BOSS), will perform these functions in ARMMS.

ARMMS Hardware

The ARMMS hardware configuration is illustrated in Figure 1. Upto four central processing units (CPU's) and four input/output processors (IOP's) will be active at any one time with up to sixteen logical memory modules available for program

instruction and data storage. A logical memory module consists of one, two, or three physical modules depending on whether the programs stored in the module are simplex, duplex, or TMR.

Upon execution of a computational task, one, two, or three CPU's will be assigned to process the task; again the number depends on the status of the task (simplex, duplex, TMR). If a task is critical enough to require redundant processing, multiple copies of the task will be stored in a logical memory module and the proper number of CPU's will execute the code in parallel with the results being compared or voted upon as they are transferred back to CM. Instructions fetched from CM are also compared or voted upon before execution. Four busses are provided for transferring addresses and data to memory and four different busses are provided for fetching instructions and data from memory. This bus structure is illustrated in Figure 2 which shows the ARMMS processor-memory interface. It is assumed that these memory modules contain plated wire components with access times of 300 ns and cycle times of 600 ns for read and 800-900 ns for write accesses to memory. The processors themselves are microprogrammable with a clock speed of 5 MHz. In order to match the memory and processor speeds, a bus speed of 10 MHz (twice the processor) is assumed.

Since the processors operate in a multiprocessing mode memory conflicts can occur. Whenever two processors require the same memory module, one will always have a higher priority. However, once one processor obtains entrance into the memory module, it cannot be preempted during that memory cycle time, even by a processor of a higher priority.

ARMMS Software

In order to provide a fast, reliable, executive, the logic in the ARMMS Control Executive System (ACES) is being kept as simple as possible. The task scheduling and facility allocation algorithms currently being considered for implementation are briefly described in the following paragraphs:

The following information about each task is required for task scheduling and resource allocation:

1. Task criticality (simplex, duplex, or TMR).
2. Task priority (the number of levels to be determined).
3. Task type (full processing, limited processing, I/O).
4. Task start time.

The resources required by a task will be determined by its criticality and type; full processing tasks require both CPU's and IOP's, limited processing tasks require only CPU's and I/O tasks require only IOP's. The number of CPU's and/or IOP's

allocated to a task is three for TMR, two for duplex and one for simplex with all the resources required by a task being assigned for the duration of its execution.

A fixed priority number is assigned to each task prior to an ARMMS mission and is used as a basis for all internal scheduling and resource allocation. Internal priority generation by ACES will be avoided if possible.

A limited processing task may be preempted during execution by a higher priority task if its preemption will yield sufficient resources to place the higher priority task in execution. Neither full processing tasks nor I/O tasks may be preempted once they begin execution because of the problems associated with the interruption of an I/O transmission.

Once ACES begins processing a task request, it cannot be interrupted or preempted by another task request regardless of the request type or the priority of the task issuing the request. Task requests originating while ACES is busy will be queued on a first-in-first-out basis. Tasks delayed due to insufficient system resources will be filed in the facilities queue on the basis of task priority.

The current ACES concept does not provide for the assigning of deadlines to tasks in order to assure execution within a certain time period. It will be the responsibility of the user to assign task priorities and task start times in such a way that task deadlines are met.

INSTRUCTION EXECUTION SIMULATION

One method that has been proposed to increase the ARMMS processor speed is to operate with an overlapped instruction fetch. It has been estimated that the additional hardware required to implement this feature is less than five percent of that which would be required for a non-overlapped processor. Instruction overlap is accomplished by allowing the processor to execute one instruction while the next instruction is being fetched from memory. Figure 3 illustrates how an instruction overlap is employed. It is apparent from this figure that the processor speed and the memory speed should be comparable in order to obtain the maximum benefit from instruction overlap. If one stream of the overlap is utilizing the processor (or memory), then the other stream is prohibited from utilizing that facility. The only exception to this rule is that the memory port becomes free at a point approximately equal to two-thirds of the memory cycle time, thus allowing the processor to transfer the next address to memory even though the memory module itself is still being utilized.

Instruction Execution Simulation Model

A simulation model which simulates the major activities involved in the execution of instructions by the ARMMS processors was developed to study the speed advantage offered by the instruction overlap features under different assumptions pertaining to the type and mix of instruction being executed, the degree of memory contention resulting from multiprocessing, the internal CPU and memory speeds, etc.

A simplified flowchart of the basic model logic is shown in Figure 4. This figure illustrates the primary and secondary paths for the overlap instruction streams. Not shown in this figure are the various queue blocks that prevent the two streams from both trying to utilize the same facility at the same time.

The model input parameters and output statistics are listed below:

Model Input Parameters

I. System Description
   - A. Number of processors
   - B. Number of memory modules
   - C. Bus Speeds
   - D. Microprogram execution time
   - E. Memory characteristics
     1. Cycle time (read and write)
     2. Time that input port is released
   - F. Instruction overlap (yes/no)

II. Software Input Characteristics
   - A. Task description
     1. Number of instructions
     2. Memory module where task is stored
   - B. Instruction description
     1. Decode time
     2. Flag if second operand is required
     3. Flag if jump type
     4. Flag if read into memory is required
     5. Execution time
   - C. Number of tasks to be executed
   - D. Relative distribution of instructions to be executed.

Model Output Statistics

I. At the completion of each task
   - A. Task number
   - B. Memory module number
   - C. Instruction Overlap (yes/no)
   - D. Processor number
   - E. Total number of instructions executed

522-137 O - 73 - 17

F. Types and frequency of instructions executed
G. Task start and end time
H. Task execution time
I. Minimum non-overlap task execution time
J. Jump-caused aborts
K. Memory and processor interference
L. Average time (in percent) gained by utilizing instruction overlap (if overlap is used).

Note that the model input parameters permit a detailed description of the characteristics of individual instructions. For the purpose of evaluating the instruction overlap technique, only 15 instructions are described in this paper; Load, Store, Conditional Jump, Unconditional Jump, Compare, Add, Subtract, Multiply, Divide, Floating Point Add, Floating Point Subtract, Floating Point Multiply, Floating Point Divide, And/Or. While this is a relatively small number of instructions, it is felt that these 15 instructions would comprise approximately 95 percent of the instructions in a typical scientific program.

Simulation Results

Some typical simulation results are shown in Figures 5 and 6. The instructions used in these simulations were those mentioned in the preceding paragraph. Within each task the mix of instructions were distributed in a fashion which approximates the Gibson mix (4).

Figure 5 shows a direct comparison between an overlapped and non-overlapped instruction stream with various degrees of memory conflict. This figure illustrates that an overlap instruction stream is about 37 percent faster than a non-overlap instruction stream, if there is no memory contention. However, as more than one processor attempts to utilize the same memory module, any advantage of instruction overlap is quickly forfeited.

Figure 6 depicts the results of running 15 tasks (6254 instructions) in a multiprocessing mode on four processors, with and without instruction overlap. Nine memory modules were assumed, and some degree of memory conflict was experienced. As can be seen from this figure, the instruction overlap processors consistently finished their tasks ahead of the non-overlap processors. This means that, with the same speed of internal components, the overlap instruction processors appear to be over 30 percent faster than the non-overlap processors.

These results show that in general, the instruction overlap feature can be expected to yield a significant improvement in processor execution speed if a moderate amount of memory contention is experienced. Only under severe memory contention does the single instruction stream processor speed approach that of the processor with instruction overlap.

EXECUTIVE SYSTEM SIMULATION

A major factor to consider in evaluating the performance of a computer system is the degree to which performance is influenced by the system executive. This effect can often be measured in an existing system by using hardware and/or software monitors to gather statistics on variations in system performance induced by changes in executive parameters or algorithms. (5),(6). However, the problem of predicting the effect on performance of an executive for a system being designed requires that some other technique, such as simulation, be employed. A simulation model was developed to study the ARMMS performance under various assumptions pertaining to facility allocation procedures task priority schemes and the time required by the BOSS hardware to execute ACES, the software component of BOSS.

Executive Simulation Model

The major model components and the principal activities simulated are illustrated in Figure 7. When simulated time reaches a task start time, that task is created, its attributes are assigned values, and it is entered into the system where it places a request to be dispatched. When this request is acknowledged by ACES (either immediately or after waiting in the EXEC queue if another request is being processed), the dispatcher module determines whether:

1. Sufficient resources are available to place the task in execution;
2. Available resources plus resources obtained by preempting lower priority tasks will meet the task resource requirements; or
3. Neither 1. or 2. is true.

The time increment T required for the dispatcher to accomplish its function is calculated as a function of a number of input parameters and the next event (initiate task after 1., preempt another task after 2., place the task in the facilities queue and process the next ACES request after 3.) is scheduled at current time plus T.

If the dispatcher module determines that either 1. or 2. is true, ACES continues processing the current task until the initiation module places the task in execution, at which time the next ACES request will be honored. Once a task begins execution, it will be interrupted only if it is a preemptable task and is preempted by a higher priority task, in which case it is placed back in the facilities queue to await another dispatch. When a task completes execution, it requests that it be terminated by the terminator module; again the task request may be

queued if the executive is busy.

The principal inputs to the module are task attributes, the number of CPU's and IOP's in the system, and timing data used to determine how long the various ACES modules require to perform their functions. Random functions are used to produce the task inter-arrival pattern and to assign the task attributes. ACES timing data are based on the criticality and type of task being processed and reflect the number of instructions which must be executed to perform the ACES logic as well as the speed of the BOSS hardware executing ACES. Module output consists of utilization statistics for the CPU's , IOP's, and the major ACES modules and statistics showing the average time spent in the system for various classes of tasks.

Baseline Simulation

A baseline simulation was performed using instruction execution estimates for the ACES timing as shown in Table 1. Workload parameter values were based upon a mission analysis profile of space missions for which the ARMMS would be a suitable onboard data processing system. (1) Instruction execution estimates for ACES were obtained from flowcharts provided by the software designer.

A number of deductions concerning the match between the workload and the system as well as the relationship between system performance and ACES can be drawn from the baseline simulation statistics shown in Table 2. Note that the dispatcher module was active almost 40% of the time, a factor influenced by the average facility queue contents of more than 46. These results suggest that a more efficient or faster dispatcher design should be explored to accommodate large queues for facilities.

Figure 8 graphically displays the average time spent in the system, in excess of the average processing time, as a function of task priority, criticality and type. Note that TMR tasks of all types encountered the longest delays in the system with full processing tasks being delayed longer than either limited processing or I/O tasks.

It is clear from the Task State Statistics (average number of tasks in queue and average number of active tasks) that the baseline system is inadequate for the workload being imposed upon it. Based upon the average task inter-arrival time of two milliseconds and the average task execution time of five milliseconds, an average of approximately 2.5 active tasks must be maintained if the system is to accommodate the workload without queue build-ups; approximately 15 percent more than the 2.21 average maintained in the baseline simulation. The

average CPU and IOP utilization maintained in the baseline indicate that perhaps changing the ACES logic and/or speeding up ACES execution would provide an adequate system without having to increase system hardware. The following paragraphs discuss a series of simulations designed to investigate system performance improvement induced by such changes in ACES.

Other Simulation Results

Some results from five simulations are presented in Table 3 where they are ranked in order of improvement over the baseline simulation; improvement being measured as increase in the average number of active tasks (or alternatively, as the decrease in lapsed time for the simulation where lapsed time was measured from the time the first run entered the system to the time the last run terminated).

Run Priority Raised After 50 Milliseconds in System--In this simulation, a simple deadline scheme was implemented in which the priority of a task was incremented by ten if it had not been dispatched within 50 milliseconds of its entry into the system. The average number of active tasks rose to 2.34, approximately a six percent increase over the baseline, indicating that perhaps a more sophisticated deadline scheme would yield additional performance improvement. It was assumed that this deadline scheme could be implemented without any increase in ACES overhead, an assumption which might be unrealistic if a more sophisticated scheme were used.

All EXEC Times Equal Zero--The purpose of this simulation was to determine whether it was possible to speed up the executive to such an extent that the workload could be accommodated, a test accomplished by exercising a model option which sets all the executive timing parameters to zero. Results from this simulation show that, while system performance may possibly be improved up to 7.7 percent by making ACES faster, this technique alone will not give the performance improvement needed.

Priority a Function of Criticality--The graphs in Figure 8 show that among the various task types the longest delays are encountered by low priority TMR tasks. In this simulation, a priority scheme was implemented to see if special consideration to TMR tasks would not only alleviate this condition, but would increase overall system performance. Task priority in this simulation was given by

$$P = P' + 10 * C$$

where

P' is baseline priority

and

C= 1 for simplex tasks
2 for duplex tasks
3 for TMR tasks

This priority scheme not only improved system performance by eight percent, but changed the task delay pattern to that shown in Figure 9. It is not surprising that with this priority scheme duplex tasks encounter the longest delays since a duplex task cannot be executed if a TMR task (now having the highest priority) is active but a simplex task can be executed.

Priority a Function of Criticality and EXEC Time Equal Zero--This simulation combined the changes incorporated in the two previous simulations and showed an improvement in performance of approximately 13 percent, providing a system which was adequate for processing the given workload. Admittedly this simulation is unrealistic with respect to the zero executive assumption, but it indicates that perhaps an optimum scheduling strategy exists which, when combined with optimum executive code, will allow the baseline hardware configuration to process the baseline workload.

Five CPU's and Five IOP's--In this simulation, an additional CPU and IOP were added to the system hardware to yield a system which is more than adequate for the baseline workload (note average queue contents and facility utilization). It is interesting to note that even in this system, which has considerable slack resources, the pattern of delay by task type established in the baseline simulation still prevails, although the magnitudes of the delays are naturally smaller. This is illustrated in Figure 10 which shows that low priority TMR tasks, especially full processing TMR tasks, still have significant delays when compared to the other task types.

The results discussed in the preceding paragraphs led to the following conclusions and recommendations pertaining to the ACES design.

1. System performance is as sensitive to the ACES logic as to the execution time of ACES.
2. Full processing stream concept should be reviewed.
3. Some special consideration may be necessary when scheduling TMR tasks.

4. If large queue build-up is anticipated in an ARMMS mission, more efficient queue search logic than is currently planned should be implemented.
5. Some sort of dynamic priority assignment or task deadline scheme may be required to assure timely completion of critical tasks.

The ACES designer concurred with recommendations 2. and 4., dropping the full processing stream concept from the ACES design and altering the queue search logic of the dispatcher to provide a more efficient and quicker method for finding the highest priority dispatchable task.

CONCLUSIONS

Simulation has proved to be an effective means for evaluating both hardware and software design concepts being considered for the Automatically Reconfigurable Modular Multiprocessor System (ARMMS). The instruction execution model demonstrated that implementing an instruction fetch overlap feature in the ARMMS processors could provide a significant improvement in processor speed. Results obtained with the ARMMS executive system simulation model identified areas in the system software where changes would improve overall system performance. These results would have been difficult to obtain without the use of simulation.

ACKNOWLEDGEMENT

The authors wish to acknowledge Dr. J.B. White (NASA-MSFC), the ARMMS project manager, for his cooperation and encouragement during the course of this work.

BIBLIOGRAPHY

1. Hughes Aircraft Company, Design of a Modular Digital Computer System, DRL4, Phase I Report, FR 72-11-450, April 15, 1972.
2. Proceedings of the ACM (SIGOPS) Workshop on System Performance Evaluation, Harvard University, April, 1971.
3. Proceedings of the ACM Symposium on the Simulation of Computer Systems, National Bureau of Standards, June, 1973. 1973.
4. Smith, J.M., "A Review and Comparison of Certain Methods of Computer Performance Evaluation", The Computer Bulletin, Vol. 11, 1968, pp. 13-18.

5. Bordsen, D.T. "UNIVAC 1108 Hardware Instrumentation System", ACM (SIGOPS) Workshop on System Performance Evaluation, Harvard University, April, 1971.
6. Schwetman, H.D., Jr., A Study of Resource Utilization and Performance Evaluation of Large-Scale Computer Systems, Technical Systems Note-12, Computation Center, University of Texas at Austin, July, 1970.

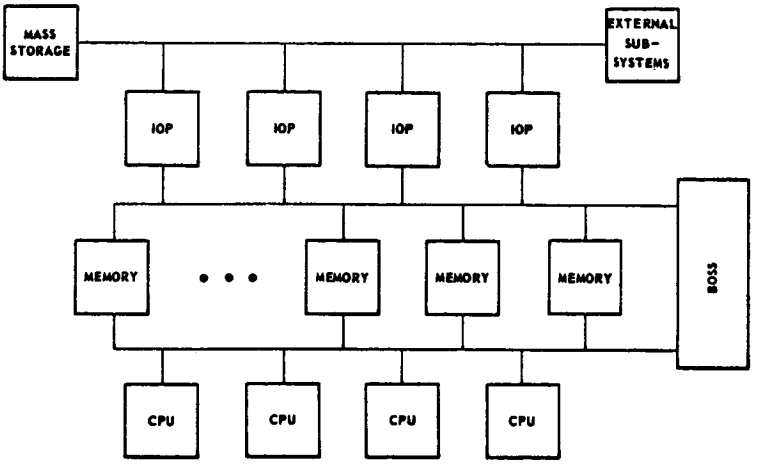

Fig. 1 – ARMMS Hardware Configuration

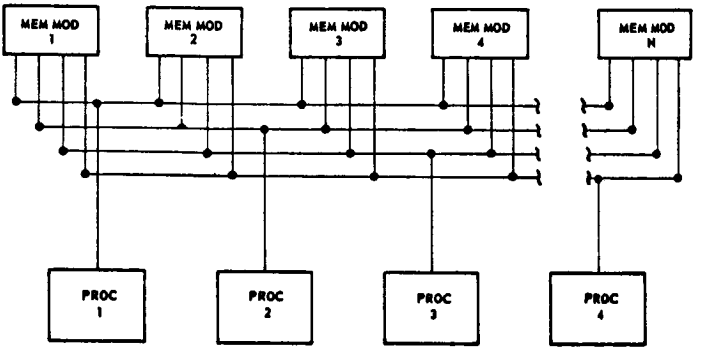

Fig. 2 – Processor– Memory Interface

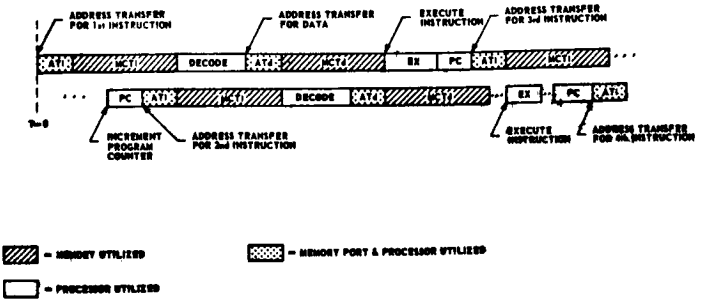

Fig. 3 – Instruction Overlap Concept

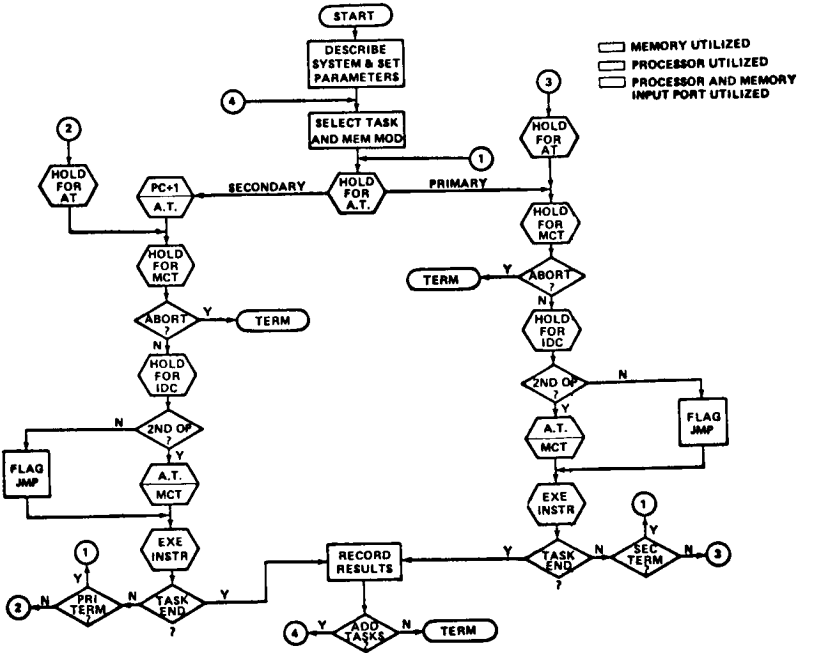

Fig. 4 – Instruction Overlap Simulation Model

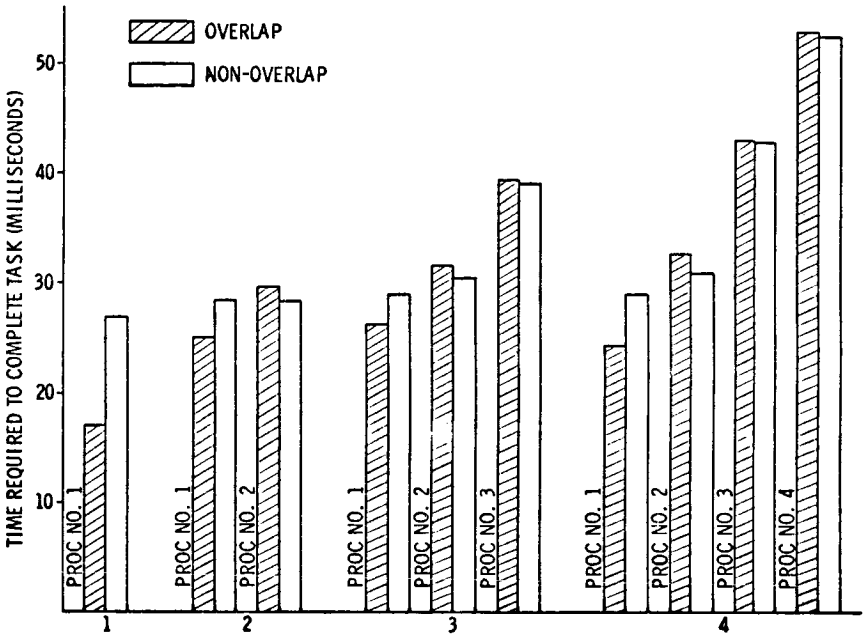

Fig. 5 – Number of Processors Contending for the Same Memory Module

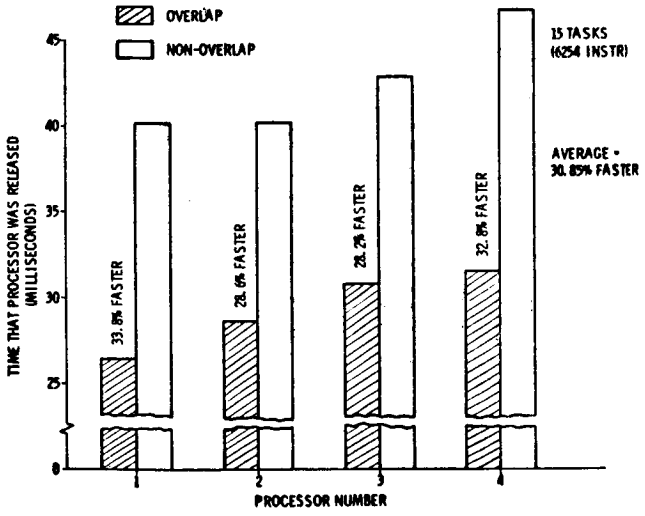

Fig. 6–

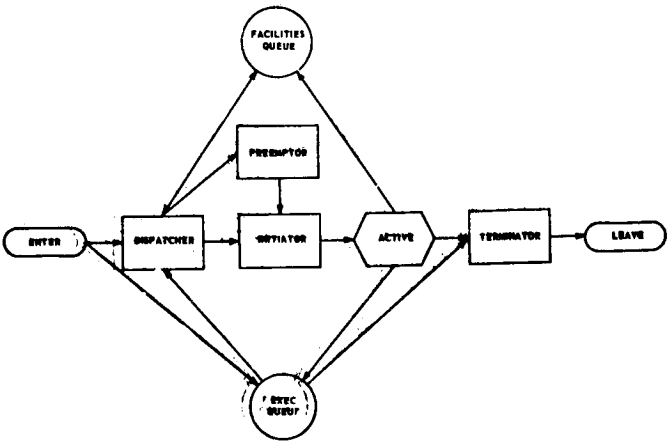

Fig. 7 – Simplified Simulator Logic

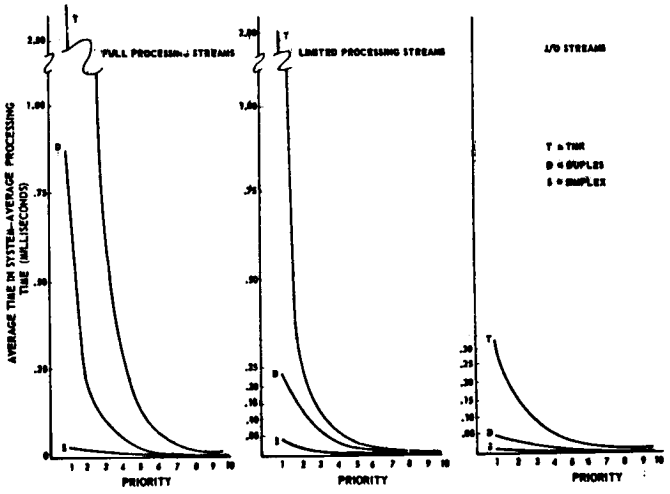

Fig. 8 – Task Wait Times for Baseline

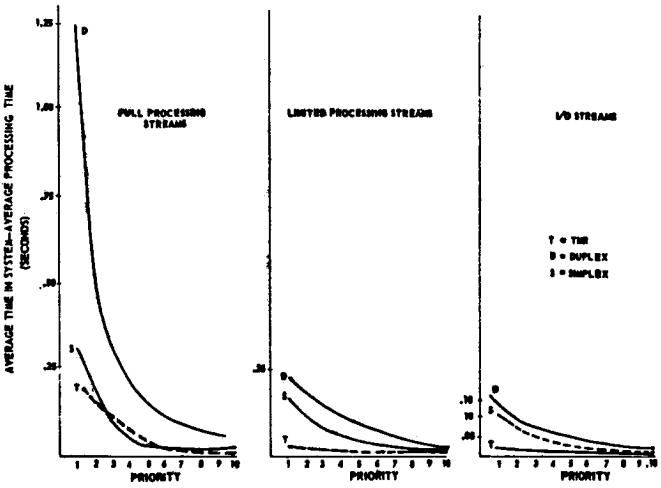

Fig. 9 – Task Wait Time with Priority Based on Criticality

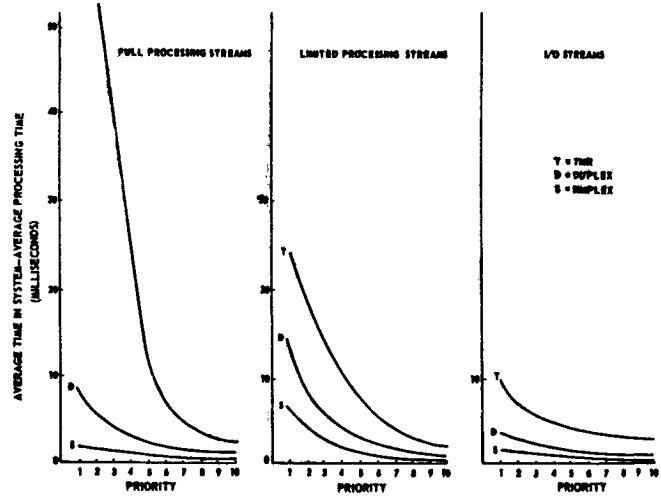

Fig. 10– Task Wait Time with 5 CPU's and 5 IOP's

PARAMETER NAME	DESCRIPTION	INSTRUCTIONS REQUIRED
FCHK	TIME TO DETERMINE WHETHER FACILITIES ARE AVAILABLE	6
PCHK	TIME TO CHECK FOR POSSIBILITY OF PREEMPTION	0–4
TABS	TIME TO SEARCH FOR AVAILABLE CONFIGURATION	24–96
TABUP	TIME TO UPDATE TABLES AFTER FACILITY ACQUISITION	30
CNFIG	TIME TO ESTABLISH CONFIGURATION	5
MNPRI	TIME TO CALCULATE LOWEST PRIORITY TASK WHICH CAN PREEMPT	0–30
INITM	TIME TO INITIATE TASK (FIRST DISPATCH)	5
RELOD	TIME TO INITIATE TASK WHICH HAS BEEN PREEMPTED	30
TERM	TIME TO TERMINATE TASK	30–60
PRTM	TIME TO PREEMPT TASK	35–65

Table 1– Executive Timing Parameters

Facility Usage Statistics

TYPE	NUMBER USERS	AVERAGE TIME/USE(MSEC)	AVERAGE UTILIZATION (%)
CPU	3649	3,773	76.0
IOP	2680	5,101	75.9

STATISTICS EXECUTIVE USAGE

MODULE	NUMBER USERS	AVERAGE TIME/USE(μ SEC)	AVERAGE UTILIZATION (%)
DISPATCHER	70,573	25	39.7
PREEMPTOR	410	130	1.4
INITIATOR	2,487	21	1.2
TERMINATOR	2,000	80	3.5

TASK STATE STATISTICS

STATE	MAXIMUM CONTENTS	NUMBER ENTRIES	AVERAGE CONTENTS	AVERAGE TIME IN STATE (MSEC)
ACTIVE	6	2487	2.21	4.024
FACILITIES QUEUE	132	68130	46.03	3.060

Table 2 –Statistical Summary of Baseline Run

RUN DESCRIPTION	EXEC TIME (SECONDS)	LAPSED TIME (SECONDS)	CPU/IOP UTILIZATION (%)	AVG. NO. ACTIVE	AVG. NO. IN QUEUE
BASELINE	2.074	4.530	76.0/75.5	2.21	55.1
RUN PRIORITY RAISED AFTER 50 MILLISECONDS IN SYSTEM	1.541	4.275	79.2/78.6	2.34	29.7
ALL EXEC TIME EQUAL ZERO	.0	4.203	78.0/78.4	2.38	23.6
PRIORITY A FUNCTION OF CRITICALITY	1.557	4.196	80.7/80.5	2.39	36.0
PRIORITY A FUNCTION OF CRITICALITY AND EXEC TIME EQUAL ZERO	.0	4.006	82.7/82.3	2.50	9.6
5 CPU'S AND 5 IOP'S	.799	3.987	67.1/66.9	2.51	2.4

Table 3 – Comparison of Simulator Run Statistics